

Exp 4

AIM: Hands on Solidity Programming Assignments for creating Smart Contracts.

Theory:

1. Primitive Data Types, Variables, Functions – pure, view

In Solidity, primitive data types form the foundation of smart contract development. Commonly used types include:

- **uint / int:** unsigned and signed integers of different sizes (e.g., uint256, int128).
- **bool:** represents logical values (true or false).
- **address:** holds a 20-byte Ethereum account address, often used for storing user accounts or contract addresses.
- **bytes / string:** store binary data or textual data.

Variables in Solidity can be **state variables** (stored on the blockchain permanently), **local variables** (temporary, created during function execution), or **global variables** (special predefined variables such as msg.sender, msg.value, and block.timestamp).

Functions allow execution of contract logic. Special types of functions include:

- **pure:** cannot read or modify blockchain state; they work only with inputs and internal computations.
- **view:** can read state variables but cannot alter them. This classification helps optimize gas usage and enforces function integrity.

2. Inputs and Outputs to Functions

Functions in Solidity can accept input arguments and return one or more output values. Inputs enable users or other contracts to pass data into the contract, while outputs make it possible to return results after computation. For example, a function can accept an amount in Ether and return whether the transfer was successful. Solidity also allows named return variables, which improve readability and debugging.

3. Visibility, Modifiers and Constructors

- **Function Visibility** defines who can access a function:
 - o **public:** available both inside and outside the contract.
 - o **private:** only accessible within the same contract.
 - o **internal:** accessible within the contract and its child contracts.
 - o **external:** can be called only by external accounts or other contracts.

- **Modifiers** are reusable code blocks that change the behavior of functions. They are often used for access control, such as restricting sensitive functions to the contract owner (onlyOwner).
- **Constructors** are special functions executed only once during contract deployment. They initialize important values, such as setting the deploying account as the owner of the contract.

3. Control Flow: if-else, loops

Control flow in Solidity is similar to traditional programming languages:

- **if-else** allows conditional decision-making in contract logic, e.g., checking if a balance is sufficient before transferring funds.
- **Loops** (for, while, do-while) enable repeated execution of code. For example, iterating through an array of users. However, loops must be used carefully, as excessive iterations increase gas consumption, potentially making the contract expensive to execute.

5. Data Structures: Arrays, Mappings, Structs, Enums

- **Arrays**: Can be fixed or dynamic and are used to store ordered lists of elements. Example: an array of addresses for registered users.
- **Mappings**: Key-value pairs that allow quick lookups. Example: mapping(address => uint) for storing balances. Unlike arrays, mappings do not support iteration.
- **Structs**: Allow grouping of related properties into a single data type, such as creating a struct Player {string name; uint score;}.
- **Enums**: Used to define a set of predefined constants, making code more readable. Example: enum Status { Pending, Active, Closed }.

6. Data Locations

Solidity uses three primary data locations for storing variables:

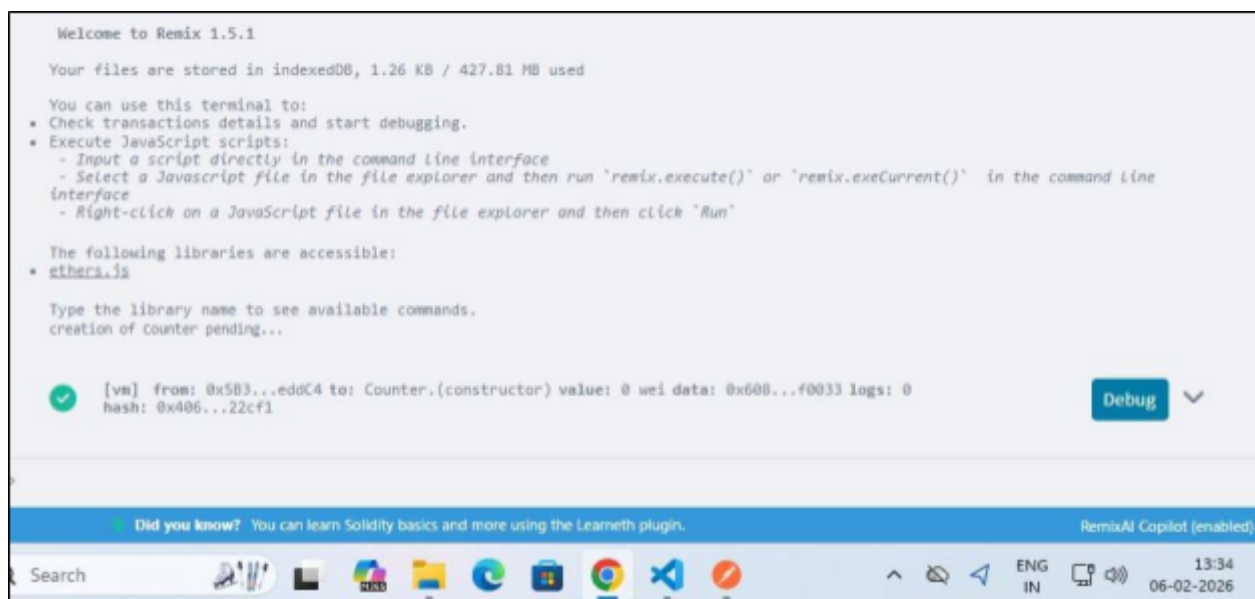
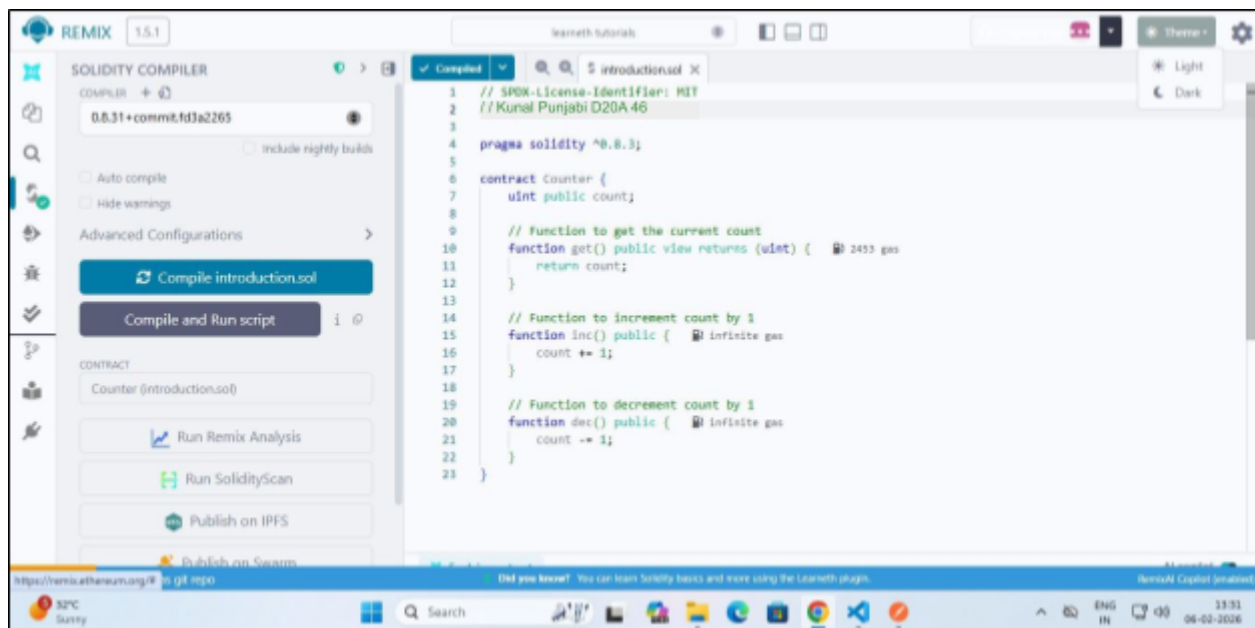
- **storage**: Data stored permanently on the blockchain. Examples: state variables.
- **memory**: Temporary data storage that exists only while a function is executing. Used for local variables and function inputs.
- **calldata**: A non-modifiable and non-persistent location used for external function parameters. It is gas-efficient compared to memory. Understanding data locations is essential, as they directly impact gas costs and performance.

7. Transactions: Ether and Wei, Gas and Gas Price, Sending Transactions

- **Ether and Wei:** Ether is the main currency in Ethereum. All values are measured in Wei, the smallest unit ($1 \text{ Ether} = 10^{18} \text{ Wei}$). This ensures high precision in financial transactions.
- **Gas and Gas Price:** Every transaction consumes gas, which represents computational effort. The gas price determines how much Ether is paid per unit of gas. A higher gas price incentivizes miners to prioritize the transaction.
- **Sending Transactions:** Transactions are used for transferring Ether or interacting with contracts. Functions like `transfer()` and `send()` are commonly used, while `call()` provides more flexibility. Each transaction requires gas, making efficiency in contract design very important.

Implementation:

Tutorial - 1 (compile)



Tutorial - 1 (inc)

✓ COUNTER AT 0XF8E...9FBEB (MEMORY)

Balance: 0 ETH

dec

inc

count

get

0: uint256: 1

Low level interactions

i

CALLDATA

Transact

Tutorial - 1 (dec)

✓ COUNTER AT 0XF8E...9FBEB (MEMORY)

Balance: 0 ETH

dec

inc

count

get

0: uint256: 0

Low level interactions

i

CALLDATA

Transact

Tutorial - 2

LEARNETH

Tutorials list

2. Basic Syntax
2 / 19

Don't worry if you didn't understand some concepts like visibility, data types, or state variables. We will look into them in the following sections.

To help you understand the code, we will link in all following sections to video tutorials from the [creator](#) of the Solidity by Example contracts.

[Watch a video tutorial on Basic Syntax.](#)

★ **Assignment**

1. Delete the HelloWorld contract and its content.
2. Create a new contract named "MyContract".
3. The contract should have a public state variable called "name" of the type string.
4. Assign the value "Alice" to your new variable.

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

basicSyntax.sol

```
1 // SPDX-License-Identifier: MIT
2 // compiler version must be greater than or equal to 0.8.3 and less than 0.9.0
3 pragma solidity ^0.8.3;
4 // @naitual Pandey D20A 37
5 // Kunal Punjabi D20A 46
6 string public name = "Alice";
7 }
```

Explain contract

AI copilot

0 ☐ Listen on all transactions

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Did you know? You can learn Solidity basics and more using the Learneth plugin.

RemixAI Copilot (enabled)

32°C Sunny

Search

ENG IN

13:46 06-02-2026

Tutorial - 3

★ **Assignment**

1. Create a new variable `newAddr` that is a `public address` and give it a value that is not the same as the available variable `addr`.
2. Create a `public` variable called `neg` that is a negative number, decide upon the type.
3. Create a new variable, `newU` that has the smallest `uint` size type and the smallest `uint` value and is `public`.

Tip: Look at the other address in the contract or search the internet for an Ethereum address.

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

newContract.sol

```
26 address public addr = 0xCA35b7d915458Ef54Aa66088dF2F44E8fA733c;
27
28
29 // Kunal Punjabi D20A 46 37
30
31 address public newAddr = 0x0000000000000000000000000000000000000000;
32 int public neg = -37;
33 uint public newU = 0;
34
35
36 // Default values
37 // Unassigned variables have a default value
38 bool public defaultBool = false;
```

Explain contract

AI copilot

0 ☐ Listen on all transactions

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Did you know? You can learn Solidity basics and more using the Learneth plugin.

RemixAI Copilot (enabled)

32°C Sunny

Search

ENG IN

13:54 06-02-2026

Tutorial - 4

A list of all Global Variables is available in the [Solidity documentation](#).

Watch video tutorials on [State Variables](#), [Local Variables](#), and [Global Variables](#).

★ Assignment

1. Create a new public state variable called `blockNumber`.
2. Inside the function `getMessage()`, assign the value of the current block number to the state variable `blockNumber`.

Tip: Look into the global variables section of the Solidity documentation to find out how to read the current block number.

Check AnswerShow answer

Next

Well done! No errors.

```
10
11
12
13
14
15
16
17
18
19
20 }

uint public blockNumber;

function getMessage() public {
    // Here are some global variables
    uint timestamp = block.timestamp; // Current block timestamp
    address sender = msg.sender; // address of the caller
}
```

ⓘ Explain contract

AI copilot

0

Listen on all transactions

Filter with transaction hash or ad...

⌕

⌵

⌵

✕

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 5

★ Assignment

1. Create a public state variable called `b` that is of type `bool` and initialize it to `true`.
2. Create a public function called `get_b` that returns the value of `b`.

Check AnswerShow answer

Next

Well done! No errors.

```
18
19
20
21
22
23 }

bool public b = true;

function get_b() public view returns (bool) {
    return b;
}
```

ⓘ Explain contract

AI copilot

0

Listen on all transactions

Filter with transaction hash or ad...

⌕

⌵

⌵

✕

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 6

★ Assignment

Create a function called `addToX2` that takes the parameter `y` and updates the state variable `x` with the sum of the parameter and the state variable `x`.

Check AnswerShow answer

Next

Well done! No errors.

```
17
18
19
20 }

function addToX2(uint y) public {
    x = x + y;
}
```

ⓘ Explain contract

AI copilot

0

Listen on all transactions

Filter with transaction hash or ad...

⌕

⌵

⌵

✕

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 7

Watch a video tutorial on Function Modifiers.

★ Assignment

1. Create a new function, `increase` in the contract. The function should take an input parameter of type `uint` and increase the value of the variable `x` by the value of the input parameter.
2. Make sure that `x` can only be increased.
3. The body of the function `increase` should be empty.

Tip: Use modifiers.

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

```
55 modifier greatthan(uint y){
56     require(y > 0, "Input parameter not greater than 0");
57 }
58
59 modifier actualincrease(uint y){
60     _;
61     x = x + y;
62 }
63 // Kunal Punjabi D20A46
64 function increase(uint y) public onlyOwner greatthan(y) actualincrease(y){
65 }
66 }
```

[Explain contract](#) AI copilot

0 ☐ Listen on all transactions

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 8

★ Assignment

Create a new function called `returnTwo` that returns the values `-2` and `true` without using a return statement.

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

```
80 function returnTwo() public pure returns(int p, bool q) {
81     p = -2;
82     q = true;
83 }
```

[Explain contract](#) AI copilot

0 ☐ Listen on all transactions

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 9

Create a new function in the `D011` contract called `testInternalVar` that returns the values of all state variables from the `base` contract that are possible to return.

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

```
67 function testInternalVar() public view returns (string memory a, string memory b){
68     return (internalVar, publicVar);
69 }
```

[Explain contract](#) AI copilot

0 ☐ Listen on all transactions

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 10

★ Assignment

Create a new function called `evenCheck` in the `base` contract:

- That takes in a `uint` as an argument.
- The function returns `true` if the argument is even, and `false` if the argument is odd.
- Use a ternary operator to return the result of the `evenCheck` function.

Tip: The modulo (%) operator produces the remainder of an integer division.

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

```
18 // }
19 // return 2;
20
21 // shorthand way to write if / else statement
22 return _x < 10 ? 1 : 2;
23 }
24 // Kunal Punjabi D20A46
25 function evenCheck(uint a) public pure returns (bool){
26     return a % 2 == 0 ? true : false;
27 }
28 }
```

[Explain contract](#) AI copilot

0 ☐ Listen on all transactions

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 11

< Tutorials list Syllabus

< 7.2 Control Flow - Loops 11 / 19 >

break

The `break` statement is used to exit a loop. In this contract, the break statement (line 14) will cause the for loop to be terminated after the sixth iteration.

[Watch a video tutorial on Loop statements.](#)

★ Assignment

1. Create a public `uint` state variable called `count` in the `Loop` contract.
2. At the end of the for loop, increment the count variable by 1.
3. Try to get the count variable to be equal to 9, but make sure you don't edit the `break` statement.

Check Answer Show answer

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Loop {
5     //Kunal Punjabi D20A 46
6     uint public count;
7     function loop() public {
8         // for loop
9         for (uint i = 0; i < 10; i++) {
10             if (i == 5) {
11                 // Skip to next iteration with continue
12                 continue;
13             }
14             if (i == 5) {
15                 // Exit loop with break
16                 break;
17             }
18             count++;
19         }
20     }
21     // while loop
```

AI copilot

note: the called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 12

< Tutorials list Syllabus

< 8.1 Data Structures - Arrays 12 / 19 >

important, then we can move the last element of the array to the place of the deleted element (line 46), or use a mapping. A mapping might be a better choice if we plan to remove elements in our data structure.

Array length

Using the length member, we can read the number of elements that are stored in an array (line 35).

[Watch a video tutorial on Arrays.](#)

★ Assignment

1. Initialize a public fixed-sized array called `arr3` with the values 0, 1, 2.
2. Make the size as small as possible.
3. Change the `getArr()` function to return the value of `arr3`.

Check Answer Show answer

Next

Well done! No errors.

```
7     uint[] public arr2 = [1, 2, 3];
8     //Kunal Punjabi D20A 46 D20A 37
9     uint[3] public arr3 = [0, 1, 2];
10    // Fixed sized array, all elements initialize to 0
11    uint[10] public myFixedSizeArr;
12
13    function get(uint i) public view returns (uint) {
14        return arr[i];
15    }
16
17    // Solidity can return the entire array.
18    // But this function should be avoided for
19    // arrays that can grow indefinitely in length.
20    function getArr() public view returns (uint[3] memory) {
21        return arr3;
22    }
23
24    function push(uint i) public {
25        // Append to array
26        // This will increase the array length by 1.
27        myFixedSizeArr.push(i);
28    }
```

AI copilot

note: the called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 13

We can use the delete operator to delete a value associated with a key, which will set it to the default value of 0. As we have seen in the arrays section.

[Watch a video tutorial on Mappings.](#)

★ Assignment

1. Create a public mapping `balances` that associates the key type `address` with the value type `uint`.
2. Change the functions `get` and `remove` to work with the mapping `balances`.
3. Change the function `set` to create a new entry to the `balances` mapping, where the key is the address of the parameter and the value is the balance associated with the address of the parameter.

Check Answer

Show answer

Next

Well done! No errors.

```
6 mapping(address => uint) public myMap;
7 // Kunal Punjabi D20A46e9 D20A 37
8 mapping(address => uint) public balances;
9
10 function get(address _addr) public view returns (uint) { 2895 gas
11     // Mapping always returns a value.
12     // If the value was never set, it will return the default value.
13     return balances[_addr];
14 }
15
16 function set(address _addr) public { 25279 gas
17     // Update the value at this address
18     balances[_addr] = _addr.balance;
19 }
20
21 function remove(address _addr) public { 5588 gas
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad...

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 14

[Watch a video tutorial on Structs.](#)

★ Assignment

Create a function `remove` that takes a `uint` as a parameter and deletes a struct member with the given index in the `todos` mapping.

Check Answer

Show answer

Next

Well done! No errors.

```
46 }
47 // Kunal Punjabi D20A46
48 function remove(uint _index) public { infinite gas
49     delete todos[_index];
50 }
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad...

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial 15

15 / 19

Another way to update the value is using the delete operator by providing the name of the enum and its member (line 35).

Removing an enum value

We can use the delete operator to delete the enum value of the variable, which means as for arrays and mappings, to set the default value to 0.

[Watch a video tutorial on Enums.](#)

★ Assignment

1. Define an enum type called `Size` with the members `S`, `M`, and `L`.
2. Initialize the variable `size` of the enum type `Size`.
3. Create a getter function `getSize()` that returns the value of the variable `size`.

Check Answer

Show answer

Next

Well done! No errors.

```
15
16
17 L
18 }
19 // Default value is the first element listed in
20 // definition of the type, in this case "Pending"
21 Status public status;
22 Size public size;
23 // Returns uint
24 // Pending - 0
25 // Shipped - 1
26 // Accepted - 2
27 // Rejected - 3
28 // Canceled - 4
29 function getSize() public view returns(Size){ 2655 gas
30     return size;
31 }
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad...

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

Tutorial - 16

★ Assignment

1. Change the value of the `myStruct` member `foo`, inside the `function f`, to 4.
2. Create a new struct `myNewStruct2` with the data location `memory` inside the `function f` and assign it the value of `myNewStruct`. Change the value of the `myNewStruct2` member `foo` to 1.
3. Create a new struct `myNewStruct3` with the data location `memory` inside the `function f` and assign it the value of `myStruct`. Change the value of the `myNewStruct3` member `foo` to 3.
4. Let the function `f` return `myStruct`, `myNewStruct2`, and `myNewStruct3`.

Tip: Make sure to create the correct return types for the function `f`.

Check AnswerShow answer

Next

Well done! No errors.

```
25 return (myStruct, myNewStruct2, myNewStruct3);
26
27
28 function f(    ) undefined gas
29     uint[] storage _arr,
30     mapping(uint => address) storage _map,
31     MyStruct storage _myStruct
32 ) internal {
33     // do something with storage variables
34 }
35
36 // You can return memory variables
37 function g(uint[] memory _arr) public returns (uint[] memory) {    infinite gas
38     // do something with memory array
39     _arr[0] = 1;
40 }
41 //Prajjwal Pandey D20A 37
42 //Kunal Punjabi D20A 46 calldata _arr) external {    468 gas
43     // do something with calldata array
44     // _arr[0] = 1;
45 }
46 }
```

Activate Windows
Go to Settings to activate Windows.

Tutorial 17

< Tutorials list Syllabus

< 10.1 Transactions - Ether and Wei 17 / 19 >

`gwei`

One `gwei` (giga-wei) is equal to 1,000,000,000 (10^9) `wei`.

`ether`

One `ether` is equal to 1,000,000,000,000,000,000 (10^{18}) `wei` (line 11).

[Watch a video tutorial on Ether and Wei.](#)

★ Assignment

1. Create a `public uint` called `oneGwei` and set it to 1 `gwei`.
2. Create a `public bool` called `isOneGwei` and set it to the result of a comparison operation between 1 `gwei` and 10^9 .

Tip: Look at how this is written for `gwei` and `ether` in the contract.

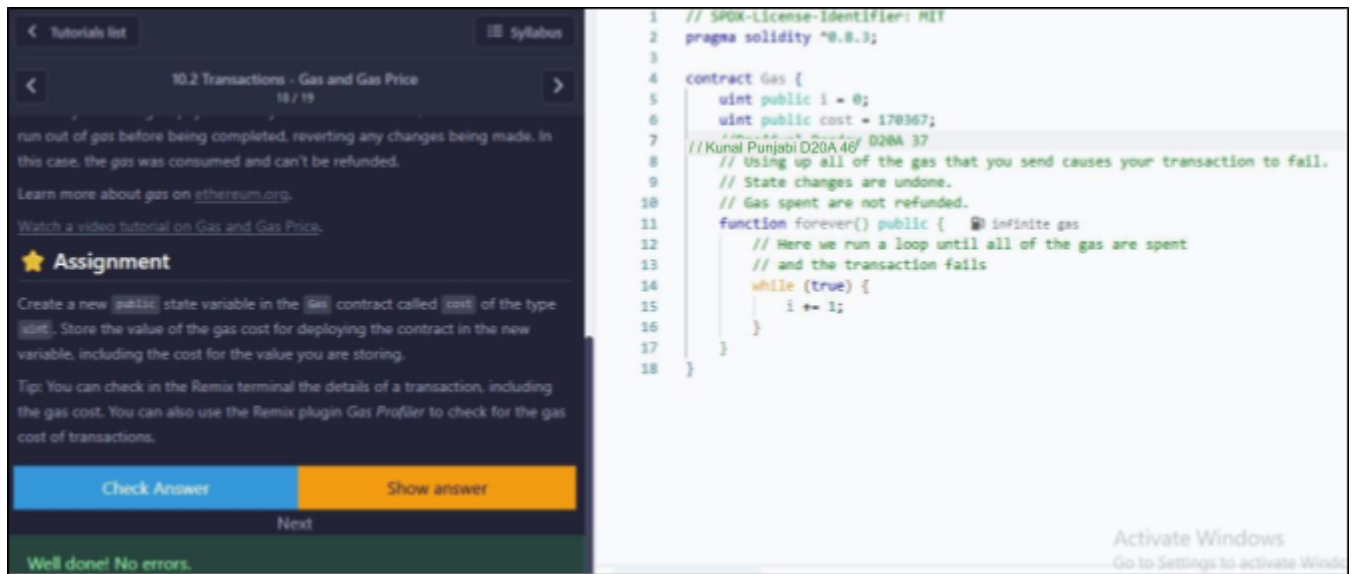
Check AnswerShow answer

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract EtherUnits {
5     uint public oneWei = 1 wei;
6     // 1 wei is equal to 1
7     bool public isOneWei = 1 wei == 1;
8
9     uint public oneEther = 1 ether;
10    // 1 ether is equal to 10^18 wei
11    bool public isOneEther = 1 ether == 1e18;
12
13    uint public oneGwei = 1 gwei;
14    // 1 ether is equal to 10^9 wei
15    bool public isOneGwei = 1 gwei == 1e9;
16 }
17 // Kunal Punjabi D20A 46
```

Tutorial 18



Assignment

Create a new `public` state variable in the `Gas` contract called `cost` of the type `uint`. Store the value of the gas cost for deploying the contract in the new variable, including the cost for the value you are storing.

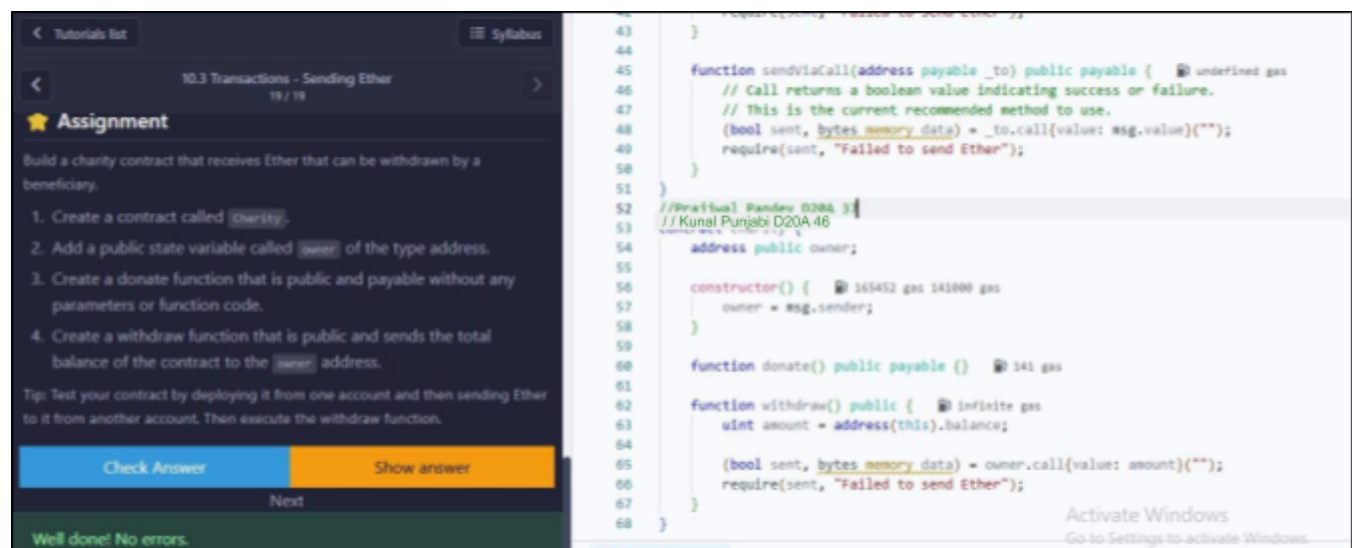
Tip: You can check in the Remix terminal the details of a transaction, including the gas cost. You can also use the Remix plugin Gas Profiler to check for the gas cost of transactions.

Code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Gas {
5     uint public i = 0;
6     uint public cost = 170367;
7     //Kunal Punjabi D20A 46/ D20A 37
8     // Using up all of the gas that you send causes your transaction to fail.
9     // State changes are undone.
10    // Gas spent are not refunded.
11    function forever() public {
12        // Here we run a loop until all of the gas are spent
13        // and the transaction fails
14        while (true) {
15            i += 1;
16        }
17    }
18 }
```

Well done! No errors.

Tutorial 19



Assignment

Build a charity contract that receives Ether that can be withdrawn by a beneficiary.

1. Create a contract called `Charity`.
2. Add a public state variable called `owner` of the type `address`.
3. Create a donate function that is public and payable without any parameters or function code.
4. Create a withdraw function that is public and sends the total balance of the contract to the `owner` address.

Tip: Test your contract by deploying it from one account and then sending Ether to it from another account. Then execute the withdraw function.

Code:

```
43 }
44
45 function sendViaCall(address payable _to) public payable {
46     // Call returns a boolean value indicating success or failure.
47     // This is the current recommended method to use.
48     (bool sent, bytes memory data) = _to.call{value: msg.value}("");
49     require(sent, "Failed to send Ether");
50 }
51
52 //Pravishal Randev D20A 31
53 //Kunal Punjabi D20A 46
54 address public owner;
55
56 constructor() {
57     owner = msg.sender;
58 }
59
60 function donate() public payable {
61 }
62
63 function withdraw() public {
64     uint amount = address(this).balance;
65     (bool sent, bytes memory data) = owner.call{value: amount}("");
66     require(sent, "Failed to send Ether");
67 }
68 }
```

Well done! No errors.

Conclusion :

Through this experiment, the fundamentals of Solidity programming were explored by completing practical assignments in the Remix IDE. Concepts such as data types, variables, functions, visibility, modifiers, constructors, control flow, data structures, and transactions were implemented and understood. The hands-on practice helped in designing, compiling, and deploying smart contracts on the Remix VM, thereby strengthening the understanding of blockchain concepts. This experiment provided a strong foundation for developing and managing smart contracts efficiently.